Curve Trees:
Practical and Transparent Zero-Knowledge Accumulators

Matteo Campanelli[1] Mathias Hall-Andersen[2]

(preliminary version)

June 13, 2022

Abstract. In this work we propose a new accumulator construction and efficient ways to prove knowledge of some element in a set without leaking anything about the element. This problem arises in several applications including privacy-preserving distributed ledgers (e.g., Zcash) and anonymous credentials. Our approaches do not require a trusted setup and significantly improve on the efficiency state of the of the art.
We introduce new techniques inspired by commit-and-prove techniques and combine shallow Merkle trees, 2-cycles of elliptic curves to obtain constructions that are highly practical. Our basic construction—which we dub *Curve Trees*—is completely transparent (does not require a trusted setup) and is based on simple standard assumptions (DLOG and Random Oracle Model). It has small proofs and commitments and very efficient proving and verification time. Curve trees can be instantiated to be efficient in practice: the commitment to a set (accumulator) is 256 bits for any set size; for a set of size 2^{32} a proof is approximately 2KB, a verifier runs in $\approx$ 160ms (easily parallelizable to $\approx$ 80ms) and a prover in $\approx$ 3.6s on an ordinary laptop.
Using our construction as a building block we can construct a simple and concretely efficient anonymous cryptocurrency with full anonymity set. We estimate the verification time to be $\approx$ 320ms (and trivially parallelizable to run in $\approx$ 160ms) or $<$ 10ms when batch-verifying multiple ($>$ 100) transactions simultaneously. Transaction sizes are $<$ 3KB. Our timings are competitive with those of the approach in Zcash Sapling and trade slightly larger proofs (proofs in Zcash are 0.2KB) for a completely transparent setup.

1 Introduction

Zero-knowledge proofs are a cryptographic primitive that allows one to prove knowledge of a secret without revealing it. In many applications the focus is on proofs that are short and with efficient running time.

One of the rising applications of zero-knowledge is in *set-membership*: given a short digest to a set S, we want to later show knowledge of a member in the set without revealing the latter. This primitive is useful in domains such as privacy-preserving distributed ledgers, anonymous broadcast, financial identities and asset governance (see, e.g., discussion in [BCF+21]).

Limitations of prior work. Our focus in this work is on solutions that are highly *practical*. That is, solutions with short proving/verification time and short proofs. While efficient solutions to zero-knowledge set-membership already exist, we argue that they have limitations. In particular, either they still have a high computational/communication cost (we elaborate in Section 1.2 where we compare to transparent polynomial commitments and ring signatures [LRR+19]) or they rely on proof systems that are *non-transparent*. The latter means that, in order for the system to be bootstrapped, it is necessary to invoke a trusted authority. This is true for example in ZCash (Sapling) [HBHW21] and in [CFH+21]. While we can partly overcome this issue by emulating the trusted authority through a large-scale MPC, this is still highly expensive, both computationally and logistically[1]. Other solutions, such as [BCF+21, CHA21], mitigate this problem by requiring a trusted setup for parameters that are reusable in other cryptographic settings (an RSA modulus). This, however, still requires invoking a trusted authority or arranging a parameter-generation ceremony [CHI+20], which may not always be viable. We then turn to solutions that are fully transparent and still very efficient.

[1] `https://z.cash/technology/paramgen/`

Our contributions. Our main contribution is a concretely efficient construction for proving private set-membership with a fully transparent setup. Specifically we design a new data structure, Curve Trees, that supports concretely small commitment to a set and where we can show set membership in zero-knowledge and with a small proof.

The design of a curve tree is simple and relies on discrete logarithm and the random oracle model (ROM) for its security. A curve tree can be described as a shallow Merkle tree where the leaves are points over an elliptic curve (and so are internal nodes). To hash, at each level we use an appropriately instantiated Pedersen hash alternating curves at each layer (we require a 2-cycle of curves). To prove membership in zero-knowledge we use commit-and-prove[2] capabilities of Bulletproofs and leverage the algebraic nature of our data structure. Our curves can be instantiated with existing ones in literature (see "Supported Curves" in Section 3.1). A curve tree has *structure-preserving* features[ACD+16] in that we never need to use any combinatorial hash (e.g., SHA) to convert representation of elements at each level or use their bit decomposition. While we focus on accumulators and set membership, our approach can straightforwardly be applied to opening of vectors rather than sets obtaining an "index-hiding" vector commitment.

Using our construction as a building block we can construct a simple and concretely efficient anonymous payment system with full anonymity set[3] and *transparent setup*. We dub this payment system $\mathbb{V}$Cash[4]. In $\mathbb{V}$Cash, the constraint system used for the zero-knowledge proof of a "spend" transaction is 20x smaller than that in ZCash Sapling (the currently deployed version of ZCash).

Concretely for two inputs/two outputs and anonymity sets of 2^{32} (like in Zcash) our confidential transactions ($\mathbb{V}$cash) require participants to compute/verify two Bulletproofs proofs of ≈ 5000 constraints each. Verifying each of the proofs in parallel (2 cores) in batches of at least 100 transactions (e.g. when verifying the validity of all transactions in a block) yields a very practical per-transaction verification time of < 10 ms[5]. Transaction sizes are < 3 KB. Our timings are competitive with those of the approach in ZCash Sapling and trade slightly larger proofs (proofs in ZCash are 0.2KB) for a completely transparent setup and simpler curve requirements.

1.1 Technical Overview

Preliminaries: elliptic-curves and SNARK-native relations In the following we assume that the reader is familiar with elliptic curves (see also Section 3.1 and notation in Section 2.1).

We informally say that a relation is "SNARK-native" to prove for a specific (SNARK) proof scheme if it is can be "naturally represented in the constraint system" (a constraint systems is a representation of a relation we aim to prove). For example, we usually consider Pedersen hashing (and commitments) to be native to Bulletproofs (instantiated in the right curve). In fact we can prove we know the scalar representation $\vec{u}$ of a group element $U = \sum_i [u_i] G_i$ through roughly $|\vec{u}|$ constraints[6]. Notice that for this operation to be actually native, each of the scalars $[u_i]$ should be elements in the scalar field of the curve to which U and the G_i-s belong to.

Starting point: shallow Merkle trees As a warm up we will ignore zero-knowledge for most of this overview and then show how to account for it. Our starting point are *shallow* Merkle trees, i.e. Merkle trees with a general branching factor $\ell \geq 2$. Let us consider a balanced tree of depth d. This has $N = \ell^d$ leaves (and it is encoding a set of an equal number of elements). We can represent this tree as labeled. Each internal node v' is labeled with the hash $H(v_1, \ldots, v_\ell)$ of the concatenation of its children; each leaf is labeled by its own value v. The root of this Merkle tree is public and represents the commitment to the set of elements. An internal node $\mathsf{rt}^{(i)}$ at level i can be seen as a root to a subtree branching from it. We denote by 0 the "lowest" level, to which the leaves belong, and by d the level to which the root belongs to (so we denote it by $\mathsf{rt}^{(d)}$).

One of the main advantages of trees with a high branching factor is that we may afford in practice a linear dependence on the depth. For a concrete vector size such as $N = 2^{32}$, we can choose $\ell = \sqrt[4]{N} = 256$ and obtain a depth $d = 4$.

The straightforward approach to opening a leaf $v^{(0)}$ in a Merkle tree opening provides the specific leaf together with the $\mathsf{rt}^{(i)}$-s, the internal nodes along the path from $v^{(0)}$ the root, and the sets $\mathsf{Siblings}(\mathsf{rt}^{(i)})$ of siblings of each

[2] In the sense of the commit-and-prove building blocks in LegoSNARK [CFQ19] and in the work by Lipmaa [Lip16].

[3] An anonymity set can be seen as the subset of existing transactions a spent transaction can be narrowed down to. We say that a protocol supports a full anonymity set then it this set consists of the whole history of transactions so far.

[4] As a reference to both ZCash and $\mathbb{V}$eksel [CHA21] from which it borrows part of its design.

[5] We use page 32 in [BBB+17] for our estimates. Our calculations are available at `https://gist.github.com/matteocam/377832d7b1f86cd0eac81149e9e65bdb`.

[6] While this is an informally defined notion, we contrast the Bulletproof example with the approach applying JubJub in Zcash Sapling. The latter is not *native* for Groth16 instantiated with BLS12-381 as it requires additional constraints for bit decompositions rather than directly describing the multiexponentiation.

$\mathsf{rt}^{(i)}$. This way, anybody can verify membership of the leaf by hashing the siblings at each level. The size of the opening certificate is roughly $\ell \cdot d$.

We would like to compress the communication complexity even further using SNARKs. However we are looking for better tradeoffs than what we can obtain by "plugging the whole tree opening inside the SNARK" (we discuss this more in the related work).

Our basic blueprint Our high-level solution stems from this insight. Instead of providing all siblings at opening time, we can just provide the internal nodes $\mathsf{rt}^{(1)}, \ldots, \mathsf{rt}^{(d)}$ together with short SNARKs $\pi^{(1)}, \ldots, \pi^{(d)}$[7]. Each proof $\pi^{(i)}$ should show "knowledge of the appropriate siblings", namely of $v_1^{(i)}, \ldots, v_\ell^{(i)}$ such that $\mathsf{rt}^{(i+1)} = \mathcal{H}(v_1^{(i)}, \ldots, v_\ell^{(i)})$ and $\mathsf{rt}^{(i)}$ is among the $v^{(i)}$-s (denoting $v^{(0)}$ by $\mathsf{rt}^{(0)}$).

Our main challenge is to keep at bay the complexity of proving (and verifying) a hash of size ℓ at each level. In order to go from this general construction to our final concrete one, there are three additional steps:

1. Going from generic hash-functions to towers of elliptic curves
2. Adding zero-knowledge (from hashes to commitments with "select-and-rerandomize")
3. Going from d to 2 proofs by moving from towers to 2-cycles of elliptic curves.

We now elaborate on each. We stress that, for sake of clarity, we somewhat simplify our explanation and leave out several optimizations we carry out in our final construction. See Section 3 and Section 4 for the actual construction.

Efficient proofs through "tower hashing" In order to obtain an efficient proof of hashing, we would like to apply a hash that is SNARK-native in the sense defined above. Our target will be applying a simple Pedersen vector hashing that is native for Bulletproofs, which is a transparent proof system. We, however, quickly run into an issue: this does not work for more than one level because it is not *structure-preserving*. An intuition about what we mean by that is: hashing at a single level would be no problem but when, when applied at multiple levels, we would need to hash the output of an earlier hash function. This would not be "native" anymore for the proof system. The problem arises because a Pedersen hash maps *field elements* to *group elements* and because we have multiple levels: a parent x of a node—an hash image—will have to be hashed again to produce its own parent—thus being part of a preimage. Node x would need to be a point in the field and in the group at the same time.

In a general version of our solution, we solve this problem by using a different hash function at each level so that we can prove it natively inside Bulletproofs at each level. In order to do this we exploit a tower of curves, with a curve at each different level. We provide more details in Section 3.1. Through this solution we can simply produce a root $\mathsf{rt}^{(i+1)}$ of ℓ children $v_1^{(i)}, \ldots, v_\ell^{(i)}$ by representing each child as a pair of coordinates $(\mathbb{x}, \mathbb{y}) \in \mathbb{F}_p$ in the base field and producing a Pedersen hash with 2ℓ generators in $\mathbb{E}_q(\mathbb{F}_p)$. We can thus produce a group element H lying in $\mathbb{E}_q(\mathbb{F}_p)$ and represent it (and its siblings) as pairs in $\mathbb{F}_q \times \mathbb{F}_q$. At the next level we can do the same using 2ℓ generators in $\mathbb{E}_r(\mathbb{F}_q)$ for another elliptic curve of order r. And so in the same way till we get to the root. In our concrete solution we will reduce this tower to a 2-cycle and use only two elliptic curves.

Adding Zero-Knowledge So far we have been concerned with solutions that do not hide which element in the vector we are opening. We actually provided the internal nodes we encounter along the opening path; therefore, in order to make our solution private, we need to modify it so to provide a *masking* of each of the internal nodes along the opening path. That is, instead of sending the actual internal node $\mathsf{rt}^{(i)}$ (the hash of its children), we let the prover sample some fresh randomness $[\rho']$ and send rerandomization (a commitment) $\mathsf{cm}^{(i)} = \mathsf{rt}^{(i)} + [\rho'] \cdot H$ (where h is a group generator). What the verifier should be shown *in zero-knowledge* at each level is a slight variation on the relation we considered before. For level i, Given $\mathsf{cm}^{(i+1)}$ and $\mathsf{cm}^{(i)}$ (respectively, the rerandomized nodes along the path at the next and current level) the verifier should be guaranteed that the prover knows some $v_1^{(i)}, \ldots, v_\ell^{(i)}, [\rho], [\rho']$ and index $j \in \{1, \ldots, \ell\}$ such that *(a)* $\mathsf{cm}^{i+1} = \mathcal{H}(v_1^{(i)}, \ldots, v_\ell^{(i)}, [\rho'])$ (the hash is again a Pedersen hash but now randomized through $[\rho']$) and *(b)* $\mathsf{cm}^{(i)} = v_j^{(i)} + [\rho]\, H$. What is important for efficiency is that relation *(a)*—a multiscalar exponentatiation—can again be proved natively as before. Relation *(b)*—a single multiplication—needs to be expressed as an arithmetic circuit. This is still relatively inexpensive for us since because it's performed once per level in a shallow tree.

Optimizing with a 2-cycle and further technical points We optimize our proof size further by applying an observation. We can move from a tower of d curves to only 2-cycle (two curves only). We can use this to reduce our communication complexity since instead of having d proofs—one per level each with a different curve—we can

[7] We anticipate that our final solution will have reduce to two proofs instead of d.

produce together two proofs—each referring to $d/2$ of the levels. See also Section 4.3 for other optimizations and technical points.

𝕍cash: transparent practical anonymous transactions. We use Curve Trees as a main building block in 𝕍cash. We defer the reader to [CHA21] for some high level ideas on the architecture (see in particular Section 1.3). The basic idea is to have the unspent coins in the systems stored as a set S of commitments in an accumulator. At the moment of transferring a coin, a user would prove in zero-knowledge that they own the coin (that is, they know the opening of some element in the set S) and provide a rerandomization of that coin (which also needs to be appropriately proved in zero-knowledge). We describe our construction at the high-level in Section 5.

1.2 Related Work

When comparing to existing approaches to zero-knowledge for set membership we focus on succinctness and prover efficiency.

Some works with transparent setup do not achieve succinctness (that is, practically short proofs and a $o(|S|)$ verification time). For example, Monero [AJ18]—or, generally, approaches based on ring signatures—have proofs linear in the set and where the verifier's running time is linear in the size of the set $|S|$.

Other approaches to accumulator with zero-knowledge properties do not involve general-purpose SNARKs. This includes for example the multilinear pairing-based polynomial commitment in [BCF+21], the seminal KZG [KZG10] and the polynomial commitments in [BMM+21]. They, however, all require knowledge-based assumptions and a trusted setup. Similar observations hold for the recent work in Caulk[ZBK+22].

Other works apply asymptotically efficient polynomial commitments with a transparent setup, but their commitment and proof size are concretely large. This is the case of Hyrax [WTs+18], where for large set sizes commitments can be $\gg 10KB$, and Dory [Lee21] where commitments are 190 bytes (6-7 times larger than ours). Proofs of single opening are also large (18 KB) in Dory, although the scheme can amortize this cost with batching (expect for very large opening batches this amortized proof size is still significantly higher than ours). The Spartan proof system has overall opening sizes, proving and verification time that are competitive with respect to ours (for sets up to approximately 2^{20} where Spartan starts to perform worse), but it has very high commitment sizes, e.g. $\geq 20KB$ for sets of size 2^{20} (625× worse than ours)[8]. Other transparent polynomial commitments include those based on Reed-Solomon IOPs [BBHR18] or on Diophantine ARguments of Knowledge (DARK) [BFS20]. As argued in [Lee21] (Section 1.1) they achieve worse concrete performances than the works above in practice.

Works that apply specialized proving techniques on accumulators in unknown-order groups: 𝕍eksel, [CHA21, CHA22, BCF+21, CFH+21]. These works obtain concretely small proofs/verifier with an efficient proving time, but require an RSA modulus (non-transparent) for their efficient instantiations[9]. While the work in [CFH+21] obtains concretely efficient proving time with a slightly larger proof size in ZCash but it require trusted setup to instantiate its proof system in addition to RSA modulus.

Using "friendlier" hash functions for Merkle trees mitigates the complexity of proving an opening. One such example is Poseidon [GKR+21]. The limitation of these solutions is that they rely on hash functions which are quite new and have not received the proper cryptanalytic scrutiny yet.

Comparing our accumulator to transparent Algebraic Merkle Trees. The most interesting comparison to our (zero-knowledge) accumulator construction is a "transparent" version of that used in ZCash. Here, for to show membership in a set we apply a specific type of Merkle tree. In it, the collision-resistant hash function we use at each level has an extra property and in particular we require it to be "algebraic" (that is it can be expressed as a polynomial in a ring). A natural choice for this—and the one applied in ZCash—is Pedersen hash. In order to prove membership we apply a zkSNARK to the opening of the Merkle tree. For this approach to be efficient we need that the group in which we compute the hash is tied to the group in which the zkSNARK "functions"[10]. ZCash uses Groth16 on curve BLS12-381 [Gro16] as a zkSNARK and a specific curve for hashing, JubJub. Nonetheless, this approach could be made transparent by applying Bulletproofs on the Ristretto curve and choosing an appropriate elliptic curve for hashing (this includes for example Jabberwock in [CHA21]). In the remainder of this comparison we refer to this way of transparently instantiating Merkle trees with Pedersen hash as $\mathsf{AlgMT}_{\mathrm{BP}}$.

[8] See [Lee21] for numbers referred in this section.

[9] In all these works we can replace the RSA group with a transparent class-group [BH01], substantially increasing their complexity. See, e.g., discussion in [DGS20]

[10] More specifically, this means that elliptic curve of the zkSNARK should be of order related to that of the definition field of the elliptic curve we use for Pedersen hashing.

We now compare the approach in our work to that in $\mathsf{AlgMT}_{\mathrm{BP}}$. Let us denote by N the set size. First we observe that asymptotically $\mathsf{AlgMT}_{\mathrm{BP}}$requires performing $\log_2(N)$ hash computations with 2 elements each (one per level of tree, hence $\log_2(N)$) inside the SNARK. Concretely, for a representative choice of $N = 2^{32}$ to ≈ 45000 constraints. Our construction, on the other hand, requires ≈ 5000 constraints (almost an order of magnitude less).

Comparing to Verkle Trees. At the *very* high-level, our approach resembles the currently explored "Verkle Tree" (VT) approach in Ethereum[11]. In both approaches an internal node represents a vector commitment to its children. The two approaches have a few substantial differences. First, that approach is currently not *structure-preserving* in the sense ours is. Each node is a commitment to a *hash* (e.g., SHA or Blake) of the children. This is required to solve a similar problem to that we approach with towers of curves. Currently Verkle Trees do no account for zero-knowledge (our focus in this work). If they did, we would also need to show in zero-knowledge that hashing the children has been performed correctly. For example, in the case of Blake this would require 20K additional constraints per level[12] (our solution, on the other hand, is in the ballpark of 5K constraint *in total*). Another difference is that the current implementation uses KZG polynomial commitments [KZG10] which requires a trusted setup.

Curve trees and Halo2. Halo2[13] is a concrete transparent (zero-knowledge) proof system that uses recursion. It is concretely efficient and it obtains recursion by going back and forth in a cycle of two curves. Halo2 and curve trees have orthogonal goals: one is a full-fledged proof system, the other can be seen as a specialized data structure (and related constraint system) for zero-knowledge for set membership. We see, however, great potential in combining the techniques in these two systems and we are currently working in this direction.

2 Preliminaries

2.1 Notation

We denote by $\mathbb{E}[\mathbb{F}]$ the elliptic curve $\mathbb{E}$ defined over the field $\mathbb{F}$, whenever clear from context we might omit the field of definition and simply write $\mathbb{E}$. Whenever possible we explicitly mark scalar elements through square brackets and group elements with upper case letters (we will occasionally break this convention if not important). We denote by $\mathfrak{M}(\mathcal{R})$ an upper bound (by construction) on the multiplicative complexity of verifying the relation $\mathcal{R}$. In practice, when estimating performance, the latter is the primary metric when proving satisfiability of arithmetic circuits.

When expressing an NP relation $R(x, w)$ we make the private witness w explicit as such but we keep the public statement x implicit. For example, in the relation $\mathcal{R}$ below

$$\mathcal{R} \coloneqq \left\{ z : y = \mathsf{SHA}(g^z) \right\}$$

the only private witness is z, while g and y are considered publicly known inputs.

2.2 Commitments

We use the following syntax for commitments:

Definition 1 (Commitments). *A commitment scheme* C *is a pair of algorithms* $(\mathsf{Setup}, \mathsf{Comm})$ *with syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$: *generates a commitment key* ck*;*
- $\mathsf{Comm}(\mathsf{ck}, m; r) \to c_m$: *produces commitment* com_m *to message* m *with randomness* r*.*

As it is standard, we call *message space* the set of of m-s for which Comm is defined and commitment space its range, $\mathsf{Rng}(\mathsf{Comm})$. We require commitments to be *perfectly hiding*—the distribution of $\mathsf{Comm}(\mathsf{ck}, m; r)$ is identical to the uniform distribution over the commitment space—and *computationally binding*—no efficient adversary can produce two pairs $(m, r), (m', r')$ such that $m \neq m'$ and $\mathsf{Comm}(\mathsf{ck}, m; r) = \mathsf{Comm}(\mathsf{ck}, m'; r')$.

Remark 1 (Rerandomizable Commitments). We will use rerandomizable commitments, i.e., endowed with an algorithm $\mathsf{Rerand}(\mathsf{ck}, \mathsf{Comm}(\mathsf{ck}, m; r)) \to (c', r')$ such that $c' = \mathsf{Comm}(\mathsf{ck}, m; r + r')$. Notice that homomorphic commitments (and thus Pedersen commitments) satisfy this property.

[11] `https://dankradfeist.de/ethereum/2021/06/18/verkle-trie-for-eth1.html`
[12] `https://github.com/zcash/zcash/issues/2258`
[13] `https://electriccoin.co/blog/explaining-halo-2/`

2.3 Accumulators

Definition 2 (Accumulator scheme). *An accumulator scheme* Acc *over universe* $\mathcal{U}_\lambda(\mathsf{Acc})$ *(where* λ *is a security parameter) consists of PPT algorithms* $\mathsf{Acc} = (\mathsf{Setup}, \mathsf{Accum}, \mathsf{PrvMem}, \mathsf{VfyMem})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pp})$ *generates public parameters* pp.
$\mathsf{Accum}(\mathsf{pp}, S) \to A$ *deterministically computes accumulator* A *for set* $S \subseteq \mathcal{U}_\lambda(\mathsf{Acc})$.
$\mathsf{PrvMem}(\mathsf{pp}, S, x) \to W$ *computes witness* W *that proves* x *is in accumulated set* S.
$\mathsf{VfyMem}(\mathsf{pp}, A, x, W) \to b \in \{0,1\}$ *verifies through witness whether* x *is in the set accumulated in* A. *We do not require parameter* x *to be in* $\mathcal{U}_\lambda(\mathsf{Acc})$ *from the syntax.*

An accumulator scheme should satisfy correctness—the accumulator works as expected—and soundness—no efficient adversary can choose a set S *and then find a witness that checks on* $\mathsf{Acc.Accum}(\mathsf{pp}, S)$ *and* $x \notin S$[14].

2.4 NIZKs

Non-Interactive Zero-Knowledge schemes (or NIZKs) require a reference string which can be either uniformly sampled (a urs), or structured (a srs). In the latter case it needs to be sampled by a trusted party. In this work we use and assume *transparent* NIZKs, i.e. whose algorithms use a reference string urs sampled uniformly.

Definition 3. *A NIZK for a relation family* $\mathfrak{R} = \{\mathfrak{R}_\lambda\}_{\lambda \in \mathbb{N}}$ *is a tuple of algorithms* $\mathsf{ZK} = (\mathsf{Prove}, \mathsf{VerProof})$ *with the following syntax:*

- $\mathsf{ZK.Prove}(\mathsf{urs}, R, x, w) \to \pi$ *takes as input a string* urs*, a relation description* R*, a statement* x *and a witness* w *such that* $R(x, w)$*; it returns a proof* π.
- $\mathsf{ZK.VerProof}(\mathsf{urs}, R, x, \pi) \to b \in \{0,1\}$ *takes as input a string* urs*, a relation description* R*, a statement* x *and a proof* π*; it accepts or rejects the proof.*

We require a NIZK to be complete, that is, for any $\lambda \in \mathbb{N}, R \in \mathfrak{R}$ *and* $(x, w) \in R$ *it holds with overwhelming probability that* $\mathsf{VerProof}(\mathsf{urs}, R, x, \pi)$ *where* $\mathsf{urs} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{poly}(\lambda)}$ *and proof* $\pi \leftarrow \mathsf{Prove}(\mathsf{urs}, R, x, w)$.

We also require knowledge-soundness and zero-knowledge to hold. Informally, the former states we can efficiently "extract" a valid witness from a proof that passes verification; the latter states that the proof leaks nothing about the witness (this is modeled through a simulator that can output a valid proof for an input in the language without knowing the witness). We use variants of these notions with certain composability properties, e.g. requiring auxiliary inputs and relation generators. For a full formal treatment of these, we refer the reader to Sections 2.2 and 2.5 in [BCFK19].

Whenever the relation family is obviously defined, we talk about a "NIZK for a relation R".

Remark 2 (Relations and Public Inputs). In the algorithms above we have both a relation R and a public input x as inputs. The reason is that in a soundness experiment, R may be constrained to be from a certain distribution on $\mathfrak{R}$ whereas x can be be chosen arbitrarily by the adversary. See for example Section 2.2 in [BCFK19]. In our constructions we often assume prover and verifier to implicitly take as input the relation description[15].

In the proof of security of our construction we require an additional property for one of our NIZKs, *simulation-extractability*. Namely, extractability should hold even with respect to an adversary that has access to simulated proofs. We refer the reader to [Gro06] for formal definitions.

Modular NIZKs through Commit-and-Prove. We use the framework for black-box modular composition of commit-and-prove NIZKs (or CP-NIZKs) in [CFQ19] and [BCFK19]. Informally a CP-NIZK is a NIZK that can efficiently prove properties of committed inputs through some commitment scheme C. Let x be a public input and c a commitment. Such a scheme can for example prove knowledge of (u, ω, r) such that $c = \mathsf{Comm}(u; r)$ and that relation $R_{\text{inner}}(x; u, \omega)$ holds. We can think of ω as a non-committed part of the witness. Besides the proof, the verifier's inputs are x and c.

In our construction we will make use of the following folklore composition to obtain efficient NIZKs from CP-NIZKs. Fixed a commitment scheme and given two CP-NIZKs $\mathsf{CP}, \mathsf{CP}'$ respectively for two "inner" relations R and R', we can prove their conjunction (for a shared witness u) $R^*(x, x', u, \omega, \omega') = R(x, u, \omega) \wedge R'(x', u, \omega')$ like this: the prover commits to u as $c \leftarrow \mathsf{Comm}(u, r)$; generates proofs π and π' from the respective schemes; it outputs combined proof $\pi^* := (c, \pi, \pi')$. The verifier checks each proof over respective inputs (x, c) and (x', c').

The following theorem (informally stated) is a direct consequence of Theorem 3.1 in [CFQ19].

[14] These definitions are standard and we refer the reader to [BBF19] for a formal treatment.
[15] This parameter is usually short.

Theorem 1 (Black-Box Composition of CP-NIZKs). *The construction above is a secure NIZK for the conjunction relation R^*.*

We can see Bulletproof [BBB+18] as a CP-NIZK since it works efficiently over an implicit commitment representation (see discussion in [CFQ19]). We use this fact in our instantiations.

3 Curve Trees as Accumulators

3.1 Towers of Elliptic Curves

We call a sequence of elliptic curves $\mathbb{E}_1(\mathbb{F}_{p_1}), \ldots, \mathbb{E}_n(\mathbb{F}_{p_n})$ a tower, iff. for $i \in [1, n-1] : p_i = |\mathbb{E}_{i+1}(\mathbb{F}_{p_{i+1}})|$; in other words the field of $\mathbb{E}_i(\mathbb{F}_{p_1})$ is the scalar field of $\mathbb{E}_{i+1}(\mathbb{F}_{p+1})$. We will generally let the field of definition be implicit to simplify notation. Towers of curves have previously been used to optimize the proving of cryptographic operations in zkSNARKs [KZM+15] [HBHW21], which will also be the application in this paper. Additionally the same techniques has been applied for recursive proofs systems [BCTV14] [BGH19]. We will not require that any of the curves are pairing friendly.

Cycles of Elliptic Curves. Of particular interest are m-cycles of elliptic curves: infinitely long towers where $\mathbb{E}_i(\mathbb{F}_{p_i}) = \mathbb{E}_{i+m}(\mathbb{F}_{p_{i+m}})$ for all i. Most commonly $m = 2$, which will be the primary case of interest in this paper as well.

3.2 The Accumulator: $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)$-Curve Tree

Our accumulation scheme is a "Curve Tree": a Curve Tree can be seen as an 'algebraically compatible' Merkle tree which uses a Pedersen commitments over $\mathbb{E}_1, \ldots, \mathbb{E}_d$ as the compression function for each level respectively. Notice that below we use a "randomness" scalar $[r]$. This will be useful for several concrete aspects of our construction described in the next section.

Definition 4 (Curve Trees). *A Curve Tree, parameterized by a branching factor ℓ and a tower of Elliptic curves $\mathbb{E}_1, \ldots, \mathbb{E}_d$ has the following recursive structure:*

Leaf: $(\ell, \mathbb{E}_1)-\mathsf{CurveTree}$**:** *A $(\ell, \mathbb{E}_1)$-Curve Tree (leaf node) is a 3-tuple $(C, 0, C)$ where $C \in \mathbb{E}_1$: a tree containing C with root C.*

Parent: $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)-\mathsf{CurveTree}$**:** *A $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)$-Curve Tree is a 3-tuple $(C, r, (T_1, \ldots, T_\ell))$, where $T_1, \ldots, T_\ell$ are $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_{d-1})$-Curve Trees, i.e.*

$$T_1 = (\hat{C}_1 = (\mathbb{x}_1, \mathbb{y}_1), \hat{r}_1, \hat{T}_1), \ldots, T_\ell = (\hat{C}_\ell = (\mathbb{x}_\ell, \mathbb{y}_\ell), \hat{r}_\ell, \hat{T}_\ell)$$

The root C of the $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)$-Curve Tree is a Pedersen commitment to the coordinates of the children:

$$C = \langle \vec{\mathbb{x}}, \vec{G}^{(d,X)} \rangle + \langle \vec{\mathbb{y}}, \vec{G}^{(d,Y)} \rangle + [r] \cdot H^{(d)}$$

For constants $\vec{G}^{(d,X)}, \vec{G}^{(d,Y)} \in \mathbb{E}_d^\ell$ and $H^{(d)} \in \mathbb{E}_d$.

3.3 Supported Curves.

Since we only require the hardness of discrete log, the techniques described in this paper are broadly applicable to any tower of curves and in particular existing cycles of elliptic curves, e.g. the "Pasta Cycle" (Pallas / Vesta curves) [Hop20], the "Tweedle Cycle" (Tweedledum / Tweedledee curves) [Hop19] or the Secp256k1 / Secq256k1 curves cycle [Poe18]. Even though our techniques do not use bilinear pairing, security of our zero-knowledge accumulator holds even in the presence of an efficiently computable bilinear map (type I, type II or type III) on one/both of the curves – enabling interoperability of our techniques with proof systems over such cycles.

4 Zero-Knowledge Set Membership in Curve Trees

In this section we describe how to prove set membership in zero knowledge for our curve trees accumulators. We use a slightly more specific version of the relation for zero knowledge for set membership in, e.g., [BCF+21, CFH+21]. Our variant (dubbed "Select-and-Rerandomize") obtains stronger unlinkability properties and is described below. We stress that our construction can be straightforwardly adapted to fit the setting in [BCF+21, CFH+21] where the commitment to the element is already given and there is no need for rerandomization.

Our final scheme achieves $O(\log n)$ communication and $O(\sqrt[d]{n})$ computation where d is the depth of the tree and n is the size of the accumulated set.

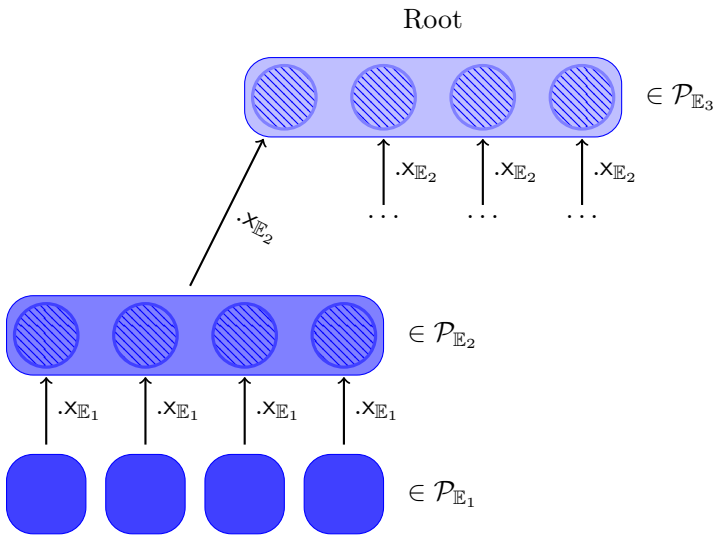

Fig. 1. Illustration of a $(4, \mathbb{E}_1, \mathbb{E}_2, \mathbb{E}_3)$-Curve Tree. Hatch pattern circles indicates that the point is represented as a field element, rounded rectangles represents Pedersen commitments to the field elements (circles) inside. Darker shades indicates lower levels in the tree.

4.1 "Select-and-Rerandomize" Accumulators

Here we define an auxiliary interface for a primitive we call a "select-and-rerandomize accumulator". Given an accumulator of committed values, I can provide you with a handle (a commitment) to a value in the accumulated set proving to you it is in the set but without revealing which one it is. We achieve this through rerandomization of commitments and zero-knowledge proofs over the set membership relation and commitment rerandomization. This primitive is natural in several settings, such as in anonymous payment systems, including the one in this work and in [CHA21]. Similar attempts at having hiding for accumulator witnesses (with different techniques) also appeared in [CFH+21].

Below we assume an accumulator scheme and an accumulated set S whose elements are (rerandomizable) commitments. We also denote by pp the concatenation of the accumulator parameters and the commitment key.

$\mathsf{SelRerand}.\mathcal{P}(\mathsf{pp}, S, c) \to (c', \pi, r')$ returns a rerandomized commitment (of $c \in S$), a proof of membership and the (auxiliary) randomness used for rerandomization.
$\mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, A, c', \pi) \to 0/1$ verifies that c' is a rerandomization of an element in the set.

Correctness: For any set S, $c \in S$ (with c an honestly generated commitment), honestly generated parameters $\mathsf{pp} = (\mathsf{pp}_{\mathrm{acc}}, \mathsf{ck})$ the following holds

$$\mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, \mathsf{Accum}(\mathsf{pp}_{\mathrm{acc}}, S), c', \pi) = 1 \ \wedge \ c' = \mathsf{Rerand}(\mathsf{ck}, c; r')$$

where $\mathsf{SelRerand}.\mathcal{P}(\mathsf{pp}, S, c) \to (c', \pi, r')$.

Security (informal): we require the proof π to be an extractable NIZK (i.e., we require knowledge soundness and zero-knowledge) for the relation below:

$$\mathcal{R}^{(\mathsf{SelectRerand})} := \left\{ (W, c, r) : \begin{array}{c} c' = \mathsf{Rerand}(\mathsf{ck}, c; r') \\ \wedge \ \mathsf{Acc.VfyMem}(\mathsf{pp}_{\mathrm{acc}}, A, c, W) \end{array} \right\}$$

whenever the parameters and the accumulator have been generated honestly.

4.2 Constructing Select-and-Rerandomize in Curve Trees

The main intuition is as follows. Observe that the leafs of a $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)$-Curve Tree are curve points on $\mathbb{E}_1$. These leafs are going to be Pedersen commitments to secret vectors. We exploit the recursive algebraic structure of our tree to enable efficient zero-knowledge proofs of knowledge of these vectors. A crucial feature of Curve Trees is that the roots (and that of every sub-tree) are rerandomizable commitments to Curve Trees: by rerandomizing the root

C as $C' \leftarrow C + [\delta] \cdot H^{(d)}$ we obtain a perfectly hiding commitment to the same set of children as the original tree, we exploit this observation to traverse the tree level-by-level using a simple zero-knowledge proof described in the next section. Another property of curve trees is that the height can dynamically increase as the number of elements in the tree grows: by using a cycle of curves the tree can grow "upward" while the leaves remain $\mathbb{E}_1$ points.

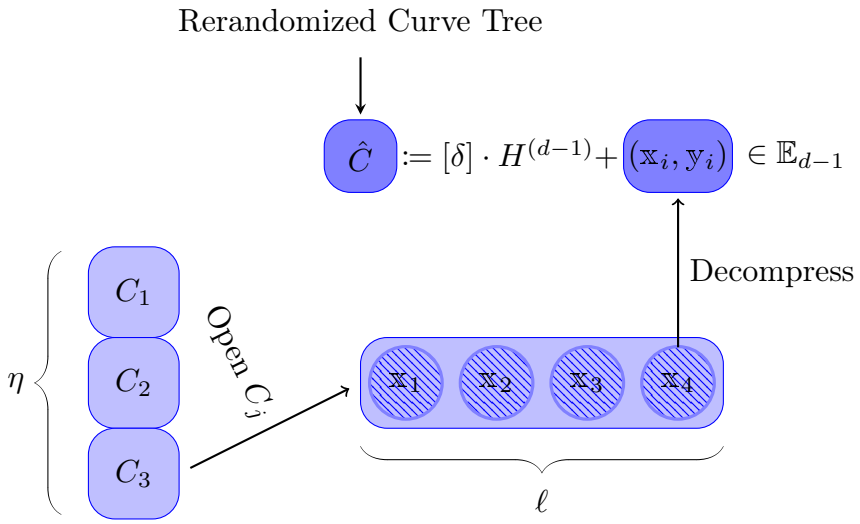

Fig. 2. Illustration of the "select and rerandomize relation". In the example illustrated above $i = 4$ and $j = 3$. Note that $\hat{C}$ is one layer deeper in the tree.

4.2.1 Single-Level Select-and-Rerandomize The central component in our construction is a simple construction for a select-and-rerandomize-like relation for a *single* level in a curve tree (we later recursively invoke this to obtain a full select-and-rerandomize in Section 4.2.2). Its underlying relation takes as input a rerandomized commitment $\hat{C}$, an alleged parent C and hidden indexes (the root of a $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)$-Curve Tree), secret index i whose semantics is "$\hat{C}$ is the i-th child of C". Practically, this is accomplished by opening the commitment C to $\vec{\mathbb{x}}, \vec{\mathbb{y}}$ using a Pedersen commit-and-prove over $\mathbb{E}_d$, then rerandomizes the i'th point $\hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d-1)} \in \mathbb{E}_{d-1}$. Slightly more formally we require a zero-knowledge argument of knowledge for the following relation:

$$\mathcal{R}^{(\mathsf{1\text{-}Lvl\text{-}CrvT})} := \left\{ (i, r, \delta, \vec{\mathbb{x}}, \vec{\mathbb{y}}) : \begin{array}{c} C = \langle \vec{\mathbb{x}}, \vec{X^{(t)}} \rangle + \langle \vec{\mathbb{y}}, \vec{Y^{(t)}} \rangle + [r] \cdot H^{(d)} \\ \wedge\ \hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d-1)} \end{array} \right\}$$

As noted the $C = \langle \vec{\mathbb{x}}, \vec{X^{(d)}} \rangle + \langle \vec{\mathbb{y}}, \vec{Y^{(d)}} \rangle + [r] \cdot H^{(d)}$ constraint can be very efficiently enforced using a commit-and-prove for Pedersen commitments (e.g. Bulletproofs or Compressed Σ-Protocol) over $\mathbb{E}_d$. While $\hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d-1)}$ requires a single fixed-based exponentiation "inside the circuit". We describe an optimized arithmetic circuit for the relation above in Appendix B.

In the appendix we describe a generalization of this relation to the "forest" case where we do not have only one root but several roots (this can be used for a flatter construction described in Remark 3).

4.2.2 Recursive $\sqrt[d]{n}$ Membership Proof. We can observe that $\mathcal{R}^{(\mathsf{1\text{-}Lvl\text{-}CrvT})}$ already yields a "one-level" select-and-rerandomize. We can apply it one level at the time and obtain a full construction. Note that in the single-level case, at a given level we can provide a (hiding) Pedersen commitment of one of the children. The latter in turn represents the root of a (sub-)Curve Tree. We can thus extend the technique presented to obtain membership proofs with $O(\sqrt[d]{n})$ prover/verifier complexity: by letting $\ell = \sqrt[d]{n}$ and using $d - 1$ individual proofs and $\eta = 1$ (except possibly for the first proof which sets $\eta = \sqrt[d]{n}$). The proofs then descends down the Curve Tree one level at a time.

Theorem 2 (informal). *The construction in Section 4.2.2 is select-and-rerandomize over curve trees as accumulators with $O(\sqrt[d]{n})$ prover/verifier complexity. This construction is transparent if we instantiate the underlying NIZK with Bulletproofs.*

$\mathsf{SelRerand}.\mathcal{P}(\mathsf{pp}, S, c^*)$

Reconstruct tree from S; let rt be its root

Let $c^{(0)}, \ldots, c^{(d)}$ be the path elements to c^* in the tree
(with $c^{(0)}$ corresponding to rt)

Let $\hat{c}^{(0)} := \mathsf{rt}$ and $r^{(0)} := 0$

for $j = 1, \ldots, d$ **do**

$(\hat{c}^{(j)}, r^{(j)}) \leftarrow \mathsf{Rerand}(\mathsf{ck}, c^{(j)})$

$\pi_j \leftarrow \mathsf{ZK.Prove}(\mathsf{pp}, \mathcal{R}^{(\mathsf{1\text{-}Lvl\text{-}CrvT})}, \hat{c}^{(j-1)}, \hat{c}^{(j)}, r^{(j)}, r^{(j-1)})$

endfor

Return $\pi^* := \left(\hat{c}^{(1)}, \ldots, \hat{c}^{(d)}, \pi^{(1)}, \ldots, \pi^{(d)}\right)$

$\mathsf{SelRerand}.\mathcal{V}(\mathsf{pp}, \mathsf{rt}, c^*, \pi^*)$

Parse π^* as $\left(\hat{c}^{(1)}, \ldots, \hat{c}^{(d)}, \pi^{(1)}, \ldots, \pi^{(d)}\right)$

Let $\hat{c}^{(0)} := \mathsf{rt}$

for $j = 1, \ldots, d$ **do**

$b_j \leftarrow \mathsf{ZK.VerProof}(\mathsf{pp}, \mathcal{R}^{(\mathsf{1\text{-}Lvl\text{-}CrvT})}, \hat{c}^{(j-1)}, \hat{c}^{(j)})$

endfor

Accept iff $\bigwedge_{j=1,\ldots,\ell} b_j = 1$

Fig. 3. Select-and-rerandomize for a $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)-\mathsf{CurveTree}$

The theorem above can be proved straightforwardly by invoking Theorem 1 at each level in the tree. Zero-knowledge follows straightforwardly by the rerandomization of the commitments and the zero-knowledge of the underlying NIZK.

4.3 Optimizations

In this section, we briefly cover some straightforward optimizations.

Merging Proofs. The approach above which necessitates a total of $d-1$ individual proofs $\pi_1, \ldots, \pi_{d-1}$. However, when using a (2) cycle of curves, all the even/odd proofs are over the same curve/field and can therefore be combined in to a single larger statement; only requiring 2 proofs in total for any d.

Point Compression / Permissible Points. In the setting where the accumulator is guaranteed to have been computed honestly (e.g. in our confidential transactions application), we can reduce the number of exponentiations during committing and the size of the witness by only committing to the $\mathbb{x}$-coordinate of the children: this remains binding by ensuring that only one of $(\mathbb{x}, \mathbb{y})$ and $(\mathbb{x}, -\mathbb{y})$ is "allowed". One common choice is to take the numerically smallest between $\mathbb{y}$ and $-\mathbb{y}$, or discriminate based upon the parity (even/odd) over $\mathbb{Z}$, however neither of these constraints can be efficiently expressed as an arithmetic circuit; instead we use a universal hash function. Let $S(v) = 1$ iff. $v \in \mathbb{F}$ is a quadratic residue (i.e. there exists $w \in \mathbb{F}$ st. $w^2 = v$) and $S(v) = 0$ otherwise. Now consider the following family of 2-universal hash functions from any field to $\{0, 1\}$:

$$\mathcal{U}_{\alpha,\beta}(v) : \mathbb{F} \to \{0, 1\}$$
$$\mathcal{U}_{\alpha,\beta}(v) \mapsto S(\alpha \cdot v + \beta)$$

Observe that the constraint $\mathcal{U}_{\alpha,\beta}(v) = 1$ can be enforced using a circuit with multiplicative complexity 1, showing $\{(w) : w^2 = (\alpha \cdot v + \beta)\}$. We exploit this to efficiently define a set of "permissible points" on $\mathbb{E}$:

$$\mathcal{P}_{\mathbb{E}} = \{(\mathbb{x}, \mathbb{y}) \mid (\mathbb{x}, \mathbb{y}) \in \mathbb{E}(\mathbb{F}_p) \wedge \mathcal{U}_{\alpha,\beta}(\mathbb{y}) = 1 \wedge \mathcal{U}_{\alpha,\beta}(-\mathbb{y}) = 0\}$$

Note that $1/4$ of the points on $\mathbb{E}$ are permissible and any $(\mathbb{x}, \mathbb{y}) \in \mathcal{P}_{\mathbb{E}}$ is uniquely defined by its $\mathbb{x}$-coordinate – this is the case for any finite field of characteristic $\notin \{2, 3\}$. Any Pedersen commitment C can be "made permissible" by simply adding H ("incrementing the randomness") until the point is permissible:

$\mathsf{MakePermissible}(C) : \mathbb{E} \to \mathcal{P}_{\mathbb{E}}$

1 : **while** $C \notin \mathcal{P}_{\mathbb{E}} : C \leftarrow C + H$

2 : **return** C

In expectation, this requires 4 curve additions and 8 square roots. By enforcing that only permissible points are added to the accumulator[16] so that the "decompression" is unique, we can reduce the complexity of $\mathcal{R}^{(\mathsf{SelectRerand})}$ slightly, instead showing:

$$\mathcal{R}^{(\mathsf{1\text{-}Lvl\text{-}CrvT})} := \left\{ (i, r, \delta, \vec{\mathbb{x}}, \mathbb{y}) : \begin{array}{c} C = \langle \vec{\mathbb{x}}, \vec{X^{(t)}} \rangle + [r] \cdot H^{(d)} \\ \wedge\ (\mathbb{x}_i, \mathbb{y}) \in \mathcal{P}_{\mathbb{E}^{(t)}} \\ \wedge\ \hat{C} = (\mathbb{x}_i, \mathbb{y}) + [\delta] \cdot H^{(d-1)} \end{array} \right\}$$

Note the $(\mathbb{x}_i, \mathbb{y}) \in \mathcal{P}_{\mathbb{E}^{(t)}}$ constraint only requires a check that $(\mathbb{x}_i, \mathbb{y}) \in \mathbb{E}^{(t)}$ in addition to $\mathcal{U}_{\alpha,\beta}(\mathbb{y}) = 1$. We include the full circuit description in Appendix B.

5 𝕍Cash: Transparent and Efficient Anonymous Payment System

In this section we informally describe our anonymous payment system, which we dub 𝕍Cash. The techniques and model here follow mostly prior work; we defer extended and formal details to the final version of this paper.

Like ZCash our construction supports the largest possible anonymity set at every transaction. On the other hand, our scheme has an additional leakage: a party S sending a transaction tx to a party R can learn when R will spend the coins received in tx (but not to whom). Only sender S can infer this.

5.1 Model

The flow of our protocol roughly follows known blueprints. We defer the reader to Section 3 and 6 and Appendix D in [CHA21, CHA22] for a formal description of a closely related model. One important difference with the formal description in 𝕍*eksel*, however, is that we do not assume parties to hold accounts. Instead, in 𝕍Cash (as in Zcash) a transaction pours pairs of input coins into pairs of output coins of equivalent value.

Intuition about the model: at any given moment in time, each party holds a certain number of coins[17] Each user is also holding a state roughly containing all the transfers occurred so far. Through the state, any user can verify the validity of each transfer and used to verify the validity of each.

Components of the model:

Setup The setup algorithm produces the initial parameters of the system. We emphasize that it does not require to be run by a trusted setup.

Pouring A sender $\mathcal{S}$ can "pour" the value of two *input coins* into two new *output coins* nullifying the input ones. The recipients of the two new coins can be distinct. It is possible for $\mathcal{S}$ itself to be one or both of the recipients. We require that the total value of input and output coins is the same.

Verifying and Processing We provide a verifying algorithm by which parties can check a transfer is valid (roughly that the sender could afford it) and processing algorithm by which parties can update their state after a transfer. Although we describe them separately for clarity they are run together in our formal construction in **??**.

5.2 Construction

Again we defer the reader to the technical overview and Section 3 in [CHA21, CHA22] for further background. Here we provide high-level details.

A transaction consist of the creation of output coins from input coins. A coin roughly consists of a commitment to its amount and other information that ensures it will be used only once and by its intended recipient.

For a transaction to be valid it must be the case that:

1. Output coins are in an appropriate non-negative range (we want to *give* money and not take it in a transaction);
2. the total value of input and output coins is the same;

[16] In case of our anonymous cryptocurrency application, this is enforced by the network of block validators: as a condition for a transaction being valid.

[17] "Holding" a coin requires knowing a certain secret key associated with the user. In this section we ignore the aspect of registering with a new key to the system, but we stress it is straightforward to add.

3. input coins "exist" and are valid themselves.

We use zero-knowledge proofs to ensure the above. The first two properties can be ensured by range proofs and homomorphic properties of Pedersen commitments respectively. The last property is where we can use our select-and-rerandomize constructions from the previous sections. All coins are stored in an accumulator. When I want to spend an input coin I can select-and-rerandomize it obtaining a rerandomized version of that coin. Now I can include that in my transaction to prove this is the rerandomization of something existing in the accumulator.

Other aspects of the system (e.g., nullifiers) can be implemented straightforwardly (e.g. through public key rerandomization) and have small impact on the concrete efficiency of our construction; we *do* include these costs in our evaluation.

5.3 Concrete Efficiency

In $\mathbb{V}$Cash, the constraint system used for the zero-knowledge proof of a "spend" transaction is 20x smaller than that in ZCash Sapling (the currently deployed version of ZCash).

Concretely for two inputs/two outputs and anonymity sets of 2^{32} (like in Zcash) our confidential transactions ($\mathbb{V}$cash) require participants to compute/verify two Bulletproofs proofs of ≈ 5000 constraints each. Verifying each of the proofs in parallel (2 cores) in batches of at least 100 transactions (e.g. when verifying the validity of all transactions in a block) yields a very practical per-transaction verification time of < 10 ms. Transaction sizes are < 3 KB. Our timings are competitive with those of the approach in ZCash Sapling and trade slightly larger proofs (proofs in ZCash are 0.2KB) for a completely transparent setup and simpler curve requirements.

References

ACD+16. Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. *Journal of Cryptology*, 29(4):833–878, October 2016.

AJ18. Kurt M. Alonso and Jordi Herrera Joancomartí. Monero - privacy in the blockchain. Cryptology ePrint Archive, Report 2018/535, 2018. `https://eprint.iacr.org/2018/535`.

BBB+17. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Report 2017/1066, 2017. `https://eprint.iacr.org/2017/1066`.

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BBF19. Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

BCF+21. Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 393–414, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

BCFK19. Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255, 2019. `https://eprint.iacr.org/2019/1255`.

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 276–294. Springer, Heidelberg, August 2014.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Heidelberg, May 2020.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. `https://eprint.iacr.org/2019/1021`.

BH01. Johannes Buchmann and Safuat Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2001.

BMM+21. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 65–97. Springer, 2021.

CFH+21. Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. Succinct zero-knowledge batch proofs for set accumulators. Cryptology ePrint Archive, Report 2021/1672, 2021. `https://ia.cr/2021/1672`.

CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019.

CHA21. Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. Cryptology ePrint Archive, Report 2021/327, 2021. `https://ia.cr/2021/327`.

CHA22. Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 652–666, 2022.

CHI+20. Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. Cryptology ePrint Archive, Report 2020/374, 2020. `https://eprint.iacr.org/2020/374`.

DGS20. Samuel Dobson, Steven D. Galbraith, and Benjamin Smith. Trustless groups of unknown order with hyperelliptic curves. Cryptology ePrint Archive, Report 2020/196, 2020. `https://eprint.iacr.org/2020/196`.

GKR+21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. pages 519–535. USENIX Association, 2021.

Gro06. Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 444–459. Springer, 2006.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

GW20. Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. https://eprint.iacr.org/2020/315.
GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.
HBHW21. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, version 2021.2.16 [nu5 proposal], 2021.
Hop19. Daira Hopwood, 2019. https://github.com/daira/tweedle.
Hop20. Daira Hopwood, 2020. https://github.com/zcash/pasta.
KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.
KZM+15. Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015. https://ia.cr/2015/1093.
Lee21. Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *Theory of Cryptography Conference*, pages 1–34. Springer, 2021.
Lip16. Helger Lipmaa. Prover-efficient commit-and-prove zero-knowledge SNARKs. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 185–206. Springer, Heidelberg, April 2016.
LRR+19. Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, November 2019.
Poe18. Andrew Poelstra, 2018. https://moderncrypto.org/mail-archive/curves/2018/000992.html.
WTs+18. Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.
ZBK+22. Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. Cryptology ePrint Archive, Paper 2022/621, 2022. https://eprint.iacr.org/2022/621.

A Forest Single-Level Select-and-Rerandomize

Here we describe a slight generalization of the relation in Section 4.2.1. Its underlying relation takes as input hidden indexes $j \in [\eta]$, $i \in [\ell]$ and η public roots of a forest of $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_d)$-Curve Trees $C_1, \ldots, C_\eta \in \mathbb{E}_d$. In this context we also assume a root $\hat{C}$ of a $(\ell, \mathbb{E}_1, \ldots, \mathbb{E}_{d-1})$-Curve Tree which is a (known[18]) rerandomization of one of the children of $C_1, \ldots, C_\eta \in \mathbb{E}_d$. Practically, this is accomplished by opening the j'th commitment C_j to $\vec{\mathbb{x}}, \vec{\mathbb{y}}$ using a Pedersen commit-and-prove over $\mathbb{E}_d$, then rerandomizes the i'th point $\hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d-1)} \in \mathbb{E}_{d-1}$. Slightly more formally we require a zero-knowledge argument of knowledge for the following relation:

$$\mathcal{R}^{(\mathsf{1\text{-}Lvl\text{-}CrvT\text{-}Frst})} := \left\{ (i, j, r, \delta, \vec{\mathbb{x}}, \vec{\mathbb{y}}) : \begin{array}{c} C_j = \langle \vec{\mathbb{x}}, \vec{X^{(t)}} \rangle + \langle \vec{\mathbb{y}}, \vec{Y^{(t)}} \rangle + [r] \cdot H^{(d)} \\ \wedge\ \hat{C} = (\mathbb{x}_i, \mathbb{y}_i) + [\delta] \cdot H^{(d-1)} \end{array} \right\}$$

Remark 3 (Simple $\sqrt{n}$ Membership Proof). Note that $\mathcal{R}^{(\mathsf{1\text{-}Lvl\text{-}CrvT\text{-}Frst})}$ immediately provides a very simple membership proof with $O(\sqrt{n})$ prover/verifier complexity using η $(\ell, \mathbb{E}_1, \mathbb{E}_2)$-Curve Trees and letting $\eta = \ell = \sqrt{n}$: to prove that a Pedersen commitment $\hat{C} = C_q + [\delta] \cdot H^{(1)} \in \mathbb{E}_1$ is a rerandomization of a commitment from the list $C_1^{(1)} = (\mathbb{x}_1^{(1)}, \mathbb{y}_1^{(1)}), \ldots, C_n^{(1)} = (\mathbb{x}_n^{(1)}, \mathbb{y}_n^{(1)}) \in \mathbb{E}_1$. Let $\eta = \ell = \sqrt{n}$ and for each $i \in [\eta]$ compute: $C_i^{(2)} = \langle \vec{\mathbb{x}}^{(1)}_{[(i-1)\cdot\ell : i\cdot\ell]}, X^{(1)} \rangle + \langle \vec{\mathbb{y}}^{(1)}_{[(i-1)\cdot\ell : i\cdot\ell]}, Y^{(1)} \rangle$ i.e. commit to each chunk of ℓ children individually. Since the size of the "select and rerandomize" relation above is $O(\eta + \ell)$, by applying a proof system with (quasi)-linear verification time we obtain a membership proof with $\widetilde{O}(\sqrt{n})$ complexity.

B Circuit Specifications

Remark 4 (Custom Gates). We keep the explication of our techniques as broadly applicable as possible: working for any elliptic curve on short wierstrass form and any commit-and-proof system for Pedersen commitments. However, the circuits in this section can be further optimized for particular curves (e.g. with non-trivial efficient endomorphisms) and proof systems (e.g. Plonk [GWC19] with custom gates for elliptic curve operations, and/or, Plookup [GW20]).

[18] In terms of extraction.

We provide all circuit specifications as Rank-1 constraints systems (R1CS): the left side of any constraint consists of a product ($\times$) of affine combinations, while the right side consists of an affine combination.

B.1 2-Set Membership

To constrain $w \in \{v_1, v_2\}$, enforce the following R1CS constraint:

$$(w - v_1) \times (w - v_2) = 0 \tag{1}$$

Most commonly $w \in \{0, 1\}$ (i.e. $v_1 = 0$ and $v_2 = 1$).

B.2 Not Zero

To enforce $v \neq 0$, introduce $\mathbb{t}_1$ and constrain:

$$\mathbb{t}_1 \times v = 1 \tag{2}$$

B.3 Curve Check

For a point $P = (\mathbb{x}, \mathbb{y}) \in \mathbb{E}(\mathbb{F})$, introduce $\mathbb{t}_1, \mathbb{t}_2$ and constraints:

$$\mathbb{x} \times \mathbb{x} = \mathbb{t}_1 \tag{3}$$
$$\mathbb{x} \times \mathbb{t}_1 = \mathbb{t}_2 \tag{4}$$
$$\mathbb{y} \times \mathbb{y} = \mathbb{t}_2 + A\mathbb{x} + B \tag{5}$$

B.4 Incomplete Curve Addition

We denote by $\dot{+}$ incomplete addition on the short Weierstrass curve $\mathbb{E}$, formally:

$$\begin{aligned}
\mathbb{E} \cup \{\bot\} \dot{+} \mathbb{E} \cup \{\bot\} &\to \mathbb{E} \cup \{\bot\} \\
\bot \dot{+} _ &\mapsto \bot \\
_ \dot{+} \bot &\mapsto \bot \\
1 \dot{+} _ &\mapsto \bot \\
_ \dot{+} 1 &\mapsto \bot \\
P \dot{+} -P &\mapsto \bot, P \in \mathbb{E} \\
P \dot{+} P &\mapsto \bot, P \in \mathbb{E} \\
P \dot{+} Q &\mapsto P + Q, P \in \mathbb{E}, Q \in \mathbb{E}, \text{ Otherwise}
\end{aligned}$$

In other words: for points $(\mathbb{x}_1, \mathbb{y}_1), (\mathbb{x}_2, \mathbb{y}_2) \in \mathbb{E}(\mathbb{F})$ the operation is undefined when $\mathbb{x}_1 = \mathbb{x}_2$ (and undefined on points not on the curve) or when one of the operands is the point at infinity. For three points (witnesses) $(\mathbb{x}_1, \mathbb{y}_1), (\mathbb{x}_2, \mathbb{y}_2), (\mathbb{x}_3, \mathbb{y}_3)$ we enforce $(\mathbb{x}_3, \mathbb{y}_3) = (\mathbb{x}_1, \mathbb{y}_1) \dot{+} (\mathbb{x}_2, \mathbb{y}_2)$, by introducing a free variable for the slope δ and the 3 constraints:

$$\delta \times (\mathbb{x}_2 - \mathbb{x}_1) = \mathbb{y}_2 - \mathbb{y}_1 \tag{6}$$
$$\delta \times (\mathbb{x}_3 - \mathbb{x}_1) = -\mathbb{y}_3 - \mathbb{y}_1 \tag{7}$$
$$\delta \times \delta = \mathbb{x}_3 + \mathbb{x}_1 + \mathbb{x}_2 \tag{8}$$

B.5 Checked Curve Addition

When exceptional cases may occur, we can check for these by enforcing distinct $\mathbb{x}$-coordinates. i.e. to enforce:

$$(\mathbb{x}_3, \mathbb{y}_3) = (\mathbb{x}_1, \mathbb{y}_1) + (\mathbb{x}_2, \mathbb{y}_2)$$

Enforce the constraints:

$$\mathbb{x}_1 \neq \mathbb{x}_2 \tag{9}$$
$$(\mathbb{x}_3, \mathbb{x}_3) = (\mathbb{x}_1, \mathbb{y}_1) \dot{+} (\mathbb{x}_2, \mathbb{y}_2) \tag{10}$$

B.6 Secret 3-Bit Lookup

An n-dimensional secret lookup in a constant table, i.e. $v = T[b_0 + 2 \cdot b_1 + 2^2 \cdot b_2]$ for secret $b_0, b_1, b_2 \in \{0,1\} \subseteq \mathbb{F}$ and $v \in \mathbb{F}^n$ with $T : \mathbb{N}_8 \to \mathbb{F}^n$. For a table $T : \mathbb{N}_8 \to \mathbb{F}$ the lookup requires 5 R1CS constants:

$$b_0 \in \{0,1\} \tag{11}$$
$$b_1 \in \{0,1\} \tag{12}$$
$$b_2 \in \{0,1\} \tag{13}$$
$$b_\& = b_1 \times b_2 \tag{14}$$

$$b_0 \times \begin{pmatrix} -T_0 \cdot b_\& + T_0 \cdot b_2 + T_0 \cdot b_1 - T_0 + T_2 \cdot b_\& \\ -T_2 \cdot b_1 + T_4 \cdot b_\& - T_4 \cdot b_2 - T_6 \cdot b_\& \\ +T_1 \cdot b_\& - T_1 \cdot b_2 - T_1 \cdot b_1 + T_1 - T_3 \cdot s_\& \\ +T_3 \cdot b_1 - T_5 \cdot b_\& + T_5 \cdot b_2 + T_7 \cdot b_\& \end{pmatrix} = \begin{matrix} T_b - T_0 \cdot b_\& + T_0 \cdot T_2 + T_0 \cdot b_1 - T_0 + T_2 \cdot b_\& \\ -T_2 \cdot b_1 + T_4 \cdot b_\& - T_4 \cdot b_2 - T_6 \cdot b_\& \end{matrix}$$

In general, for tables $T : \mathbb{N}_8 \to \mathbb{F}^n$ the technique above requires $4+n$ constaints: repeating the last constaint for each additional coordinate.

B.7 Circuit for Fixed-Base Exponentiation

Abusing notation, we write $(\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = (\mathbb{x}, \mathbb{y}) \dot{+} T$ for the constraint: $(\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = (\mathbb{x}, \mathbb{y}) \dot{+} (\hat{\mathbb{x}}, \hat{\mathbb{y}})$ and $(\hat{\mathbb{x}}, \hat{\mathbb{y}}) \in T$. Multiplying a constant curve point by a secret scalar is implemented by decomposing the scalar into 3-bit windows (b_0, b_1, b_2) and defining the tables T st. the exceptional cases does not occur (except for the last table – where we use the checked version). Let $m = \lfloor \lambda/3 \rfloor + 1$, for for $i \in 1, \ldots, m-1$, define the table T_i as:

$$T_i = \left\{ \left[j \cdot 2^{3 \cdot i} + 2^{3 \cdot (i+1)} \right] \cdot H \;\middle|\; j \in 0, \ldots, 2^3 - 1 \right\}$$

Define T_m as follows:

$$T_m = \left\{ \left[j \cdot 2^{3 \cdot m} - \sum_{i=1}^{m-1} 2^{3 \cdot i} \right] \cdot H \;\middle|\; j \in 0, \ldots, 2^3 - 1 \right\}$$

To enforce $(\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = [r] \cdot H + (\mathbb{x}, \mathbb{y})$, we express it as:

$$(\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = \mathsf{Rerand}(\mathbb{x}, \mathbb{y}) \coloneqq (\tilde{\mathbb{x}}, \tilde{\mathbb{y}}) = (\mathbb{x}, \mathbb{y}) + (T_m + (T_{m-1} \dot{+} (T_{m-2} \dot{+} (\ldots)))) \tag{15}$$

And decompose with witness $r \in \mathbb{Z}_{|\langle H \rangle|}$ as

$$r = \sum_i v_i \cdot 2^{3i}$$

B.8 Range Check

A range check for $v \in [0, 2^i)$ requires i constraints:

$$\forall\ b_i \in \{0,1\} \tag{16}$$
$$v = \sum_i 2^i \cdot b_i \tag{17}$$

B.9 Selection

Selecting a single secret entry (hidden index) from a secret vector:

$$\mathbb{x} = \mathsf{Select}(\vec{\mathbb{X}}) \coloneqq \{(\mathbb{x}, \vec{I}, \vec{\mathbb{X}}) : \forall j : I_j \in \{0,1\} \wedge \mathbb{x} = \sum_{j=1} \mathbb{X}_i \cdot I_i \wedge 1 = \sum_{j=1} I_j\}$$

B.10 Spending Relation

The spending relation consists of a relation for each curve in the tower of curves (see Section 3.1). Below we describe the spending relation for a two cycle of curves $(\mathbb{E}, \hat{\mathbb{E}})$. The spending relation for the commit-and-prove over $\mathbb{E}$:

$$
\begin{aligned}
\mathsf{Spend}(\mathsf{v}, \mathrm{pk}, C_1) \coloneqq \Big\{ & ((\vec{\mathbb{X}}_i, \hat{\mathbb{x}}_i, \hat{\mathbb{y}}_i)_{i \in 1,\ldots,d/2}, \mathsf{v}) : \\
& \mathsf{Root} = C_1 = (\mathbb{x}_1, \mathbb{y}_1) = \mathsf{Commit}(\vec{\mathbb{X}}_1) && /\!/ \text{ Open commitment to vector of } \mathbb{x}\text{-coordinates} && (18) \\
& \hat{\mathbb{x}}_1 = \mathsf{Select}(\vec{\mathbb{X}}_1) && /\!/ \text{ Select a coordinate.} && (19) \\
& (\hat{\mathbb{x}}_1, \hat{\mathbb{y}}_1) \in \mathcal{P}_{\hat{\mathbb{E}}} && /\!/ \text{ Decompress to "permissible" point.} && (20) \\
& \hat{C}_1 = (\hat{\mathbb{x}}'_1, \hat{\mathbb{y}}'_1) = \mathsf{Rerand}(\hat{\mathbb{x}}_1, \hat{\mathbb{y}}_1) && /\!/ \text{ Rerandomize inner commitment.} && (21) \\
& \qquad \vdots \\
& C_{d/2-1} = (\mathbb{x}_{d/2-1}, \mathbb{y}_{d/2-1}) = \mathsf{Commit}(\vec{\mathbb{X}_{d/2-1}}) && && (22) \\
& \hat{\mathbb{x}}_{d/2-1} = \mathsf{Select}(\vec{\mathbb{X}_{d/2-1}}) && && (23) \\
& (\hat{\mathbb{x}}_{d/2-1}, \hat{\mathbb{y}}_{d/2-1}) \in \mathcal{P}_{\hat{\mathbb{E}}} && && (24) \\
& \hat{C}_{d/2-1} = (\hat{\mathbb{x}}'_{d/2-1}, \hat{\mathbb{y}}'_{d/2-1}) = \mathsf{Rerand}(\hat{\mathbb{x}}_{d/2-1}, \hat{\mathbb{y}}_{d/2-1}) && && (25) \\
& \hat{C}_{d/2} = \mathsf{Commit}(\mathrm{pk}, \mathsf{v}) && /\!/ \text{ Partially open the final commitment. } \mathsf{v} \text{ is hidden.} \Big\} && (26)
\end{aligned}
$$

Note that pk is a part of the statement (public input). The spending relation for the commit-and-prove over $\hat{\mathbb{E}}$:

$$
\begin{aligned}
\Big\{ & ((\vec{\mathbb{X}}_i, \mathbb{x}_i, \mathbb{y}_i)_{i \in 1,\ldots,d/2}) : \\
& \hat{C}_1 = (\hat{\mathbb{x}}_1, \hat{\mathbb{y}}_1) = \mathsf{Commit}(\vec{\mathbb{X}}_1) && /\!/ \text{ Open commitment to vector of } \mathbb{x}\text{-coordinates} && (27) \\
& \mathbb{x}_1 = \mathsf{Select}(\vec{\mathbb{X}}_1) && /\!/ \text{ Select a coordinate.} && (28) \\
& (\mathbb{x}_1, \mathbb{y}_1) \in \mathcal{P}_{\mathbb{E}} && /\!/ \text{ Decompress to "permissible" point.} && (29) \\
& C_1 = (\mathbb{x}'_1, \mathbb{y}'_1) = \mathsf{Rerand}(\mathbb{x}_1, \mathbb{y}_1) && /\!/ \text{ Rerandomize inner commitment.} && (30) \\
& \qquad \vdots \\
& \hat{C}_{d/2} = (\hat{\mathbb{x}}_{d/2}, \hat{\mathbb{y}}_{d/2}) = \mathsf{Commit}(\vec{\mathbb{X}_{d/2}}) && && (31) \\
& \mathbb{x}_{d/2} = \mathsf{Select}(\vec{\mathbb{X}_{d/2}}) && && (32) \\
& (\mathbb{x}_{d/2}, \mathbb{y}_{d/2}) \in \mathcal{P}_{\mathbb{E}} && && (33) \\
& C_{d/2} = (\mathbb{x}'_{d/2}, \mathbb{y}'_{d/2}) = \mathsf{Rerand}(\mathbb{x}_{d/2}, \mathbb{y}_{d/2}) \Big\} && && (34)
\end{aligned}
$$

The statement (public input) is defined by $x = (\mathrm{pk}, (C_i, \hat{C}_i)_{i \in 1,\ldots,d/2})$. Note that $C_1 = \mathsf{Root}$ is the root of the Curve Tree containing all prior transaction outputs.

B.11 Minting Relation

The mintining relation is only over $\mathbb{E}$ and consists of a range check:

$$
\begin{aligned}
\mathsf{Mint}(\mathsf{v}, \mathrm{pk}, C) \coloneqq \{ & (\mathsf{v}, \mathrm{pk}) : && && (35) \\
& C = \mathsf{Commit}(\mathsf{v}, \mathrm{pk}) && && (36) \\
& \mathsf{v} \in [0, 2^{64}) \} && && (37)
\end{aligned}
$$

Note that the pk is part of the witness.

B.12 2-to-2 UTXO Relation

The 2-to-2 UTXO relation consists of two instances of the spending relation, two instances of the minting relation and a check that the balances (minus a transaction fee) match.

$$\begin{aligned}
\mathsf{UTXO}(\mathsf{Root}, C_{\mathsf{out1}}, C_{\mathsf{out2}}) \coloneqq \Big\{ & (\mathsf{v}_{\mathsf{in1}}, \mathsf{v}_{\mathsf{in2}}, \mathsf{v}_{\mathsf{out1}}, \mathsf{v}_{\mathsf{out2}}, \mathrm{pk}_{\mathsf{out1}}, \mathrm{pk}_{\mathsf{out2}}) : && (38)\\
& \mathsf{Mint}(\mathsf{v}_{\mathsf{out1}}, \mathrm{pk}_{\mathsf{out1}}, C_{\mathsf{out1}}) && (39)\\
& \mathsf{Mint}(\mathsf{v}_{\mathsf{out2}}, \mathrm{pk}_{\mathsf{out2}}, C_{\mathsf{out2}}) && (40)\\
& \mathsf{Spend}(\mathsf{v}_{\mathsf{in1}}, \mathrm{pk}_{\mathsf{in1}}, \mathsf{Root}) && (41)\\
& \mathsf{Spend}(\mathsf{v}_{\mathsf{in2}}, \mathrm{pk}_{\mathsf{in2}}, \mathsf{Root}) && (42)\\
& \mathsf{v}_{\mathsf{in1}} + \mathsf{v}_{\mathsf{in2}} = \mathsf{v}_{\mathsf{out1}} + \mathsf{v}_{\mathsf{out2}} + \mathsf{v}_{\mathsf{fee}} \Big\} && (43)
\end{aligned}$$

Note that $\mathrm{pk}_{\mathsf{in1}}$ and $\mathrm{pk}_{\mathsf{in2}}$ are public, while $\mathrm{pk}_{\mathsf{out1}}$ and $\mathrm{pk}_{\mathsf{out2}}$ are part of the witness.